Санкт-Петербургский государственный университет
Математико-механический факультет
Кафедра исследования операций
Направление Математика

Большакова Елена Андреевна

Свойство бисвязности ориентированного графа

Выпускная квалификационная работа

Научный руководитель:
Доктор физ.-мат. наук, профессор
Романовский Иосиф Владимирович

Рецензент:
Ведущий научный сотрудник СПб ЭМИ РАН,
доктор физ.-мат. наук Крепс Виктория Леонидовна

Санкт-Петербург
2017

Saint-Petersburg State University
The Faculty of Mathematics and Mechanics
Subdepartment of Operations Research
Speciality Mathematics

Bolshakova Elena Andreevna

Property of bicconnectivity of a directed graph

Graduation Project

Scientific Supervisor:
Doctor of Physics and Mathematics, professor
I.V. Romanovskiy

Reviewer:
Leading Researcher of the EMI RAS in Saint-Petersburg,
Doctor of Physics and Mathematics V.L. Kreps

Saint-Petersburg
2017

Содержание

Введение

В данной дипломной работе рассматривается метод нахождения компонент сильной связности SCC ориентированного графа. Компоненты сильной связности SCC – это подграфы (максимальные по включению) такие, что любые две вершины, принадлежащие подграфу, достижимы друг из друга. Задача на нахождение компонент встречалась в литературе и описана в книге А. Ахо, Д. Хопкрофта и Д. Ульмана «Структуры данных и алгоритмы».

Мне было поручено реализовать представленный в этой книге алгоритм Косарайю (1983 г.). Что и было сделано.

Алгоритм был написан на языке c++ и проверен на десятке примеров графов разной размерности.

Графо-теоретические понятия

Граф – это совокупность непустого множества вершин и наборов ребер (связей между вершинами).

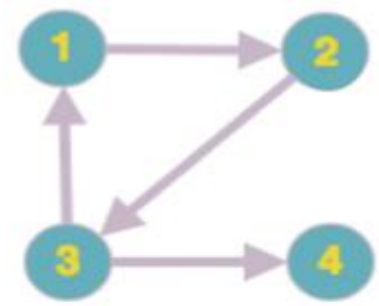
Ориентированный граф (сокращенно *орграф*) G – это упорядоченная пара $G := (V, E)$, где V – множество вершин (V-vertex), E – множество (упорядоченных) пар различных вершин, называемых дугами или ориентированными ребрами (E-edge).

Дуга – это упорядоченная пара вершин (v, w), где вершину v называют началом, а w – концом дуги.

Можно сказать, что дуга $v \mapsto w$ ведет от вершины v к вершине w.

Другими словами, орграф – это такой граф, у которого все ребра имеют направление.

Орграф, полученный из простого графа ориентацией ребер, называется *направленным*.

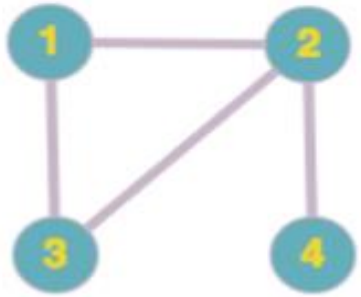


Ориентированный граф

Directed Graph

$$V = \{1,2,3,4\}$$

$$E = \{(1,2)(3,4)(3,1)(2,3)\}$$



Неориентированный граф

Undirected Graph

Для неориентированного графа:

Компонента связности – это максимальный подграф, такой, что для любой его пары вершин существует цепь.

Связный граф – это граф, содержащий ровно одну компоненту связности.

Ациклический граф – граф без циклов.

Дерево – связный ациклический граф.

Шарнир (точка сочленения / разделяющая вершина) – вершина, удаление которой увеличивает число компонент связности.

Мост – ребро, удаление которого увеличивает число компонент связности.

Поиск в глубину DFS

Поиск в глубину (DFS – depth first search) – метод обхода графа.

Поиск в глубину представляет особый метод, который является одним из основных алгоритмов на графах. Этот метод служит для решения задач поиска связности, а также множества других задач, связанных с обработкой графов. Данный метод очень значим, так как именно он послужил основой для разработки ряда других быстрый алгоритмов.

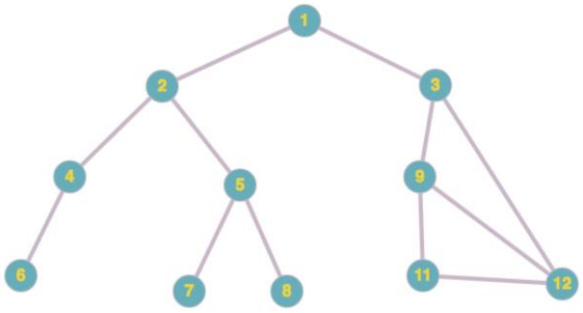
Суть данного метода заключается в том, чтобы двигаться «вглубь» до тех пор, пока это возможно. Обход вершин графа происходит по принципу: если из текущей вершины есть ребра, ведущие в непройденные вершины, то идем туда, иначе возвращаемся назад, и обход продолжается для оставшегося графа.

Рассмотрим подробнее алгоритм DFS пошагово.

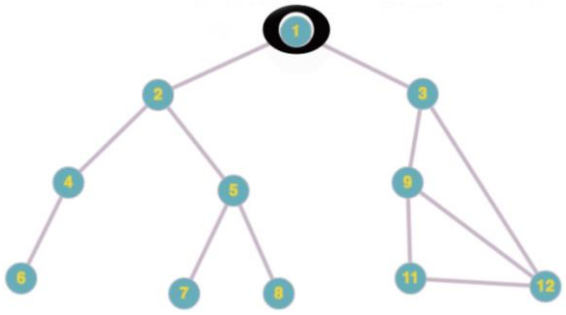
1. Для начала должна быть выбрана начальная вершина v графа G, которая, в свою очередь, автоматически помечается и отмечается как пройденная.
2. На втором шаге рекурсивно вызывается поиск в глубину DFS для каждой непомеченной вершины, которая смежна с v.
3. Продолжается этот поиск до тех пор, пока все вершины, достижимые из v, не будут помечены.
4. В случае, если на каком-то шаге обхода поиск закончился, но, тем не менее, некоторые вершины так и остались непомеченными, то должна быть выбрана произвольная непомеченная вершина, и поиск должен быть повторен.
5. Процесс поиска закончен только тогда, когда все вершины графа G будут помечены.

Для более точного понимания DFS рассмотрим пример.

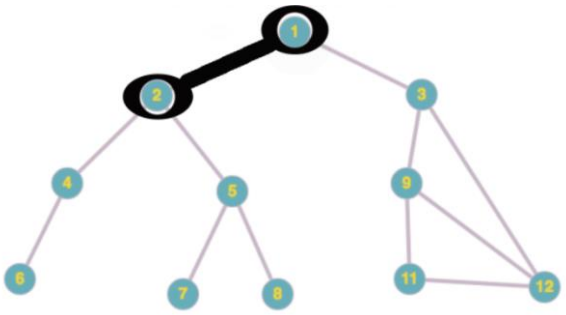
Нам дан граф с 12 вершинами.



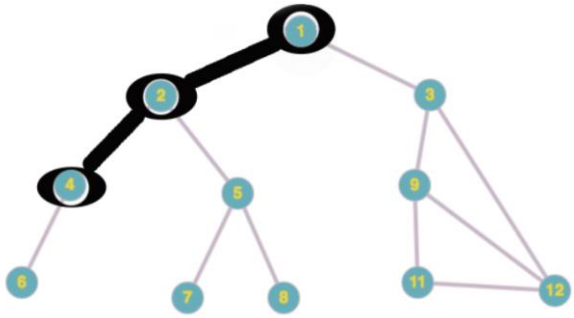
Берем вершину **1** и помечаем ее.



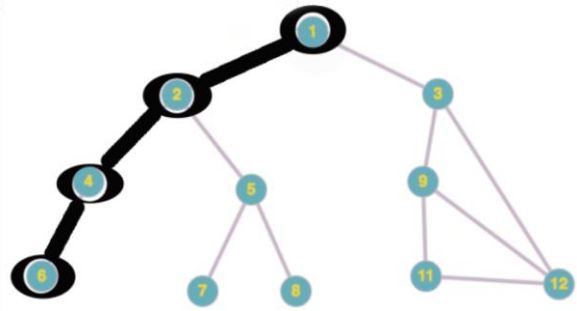
Далее ищем смежные ней вершины, двигаясь «вглубь». Помечаем таким образом **2**.



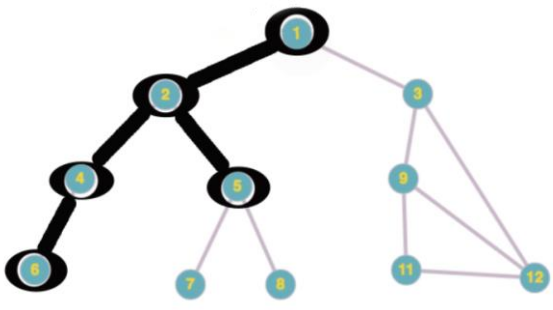
Так же, двигаясь вглубь графа, находим вершину **4**.



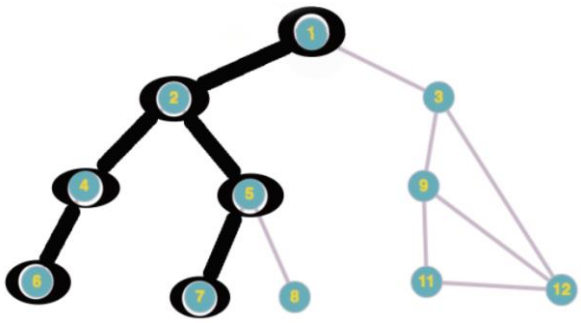
И, наконец, видим вершину **6**.



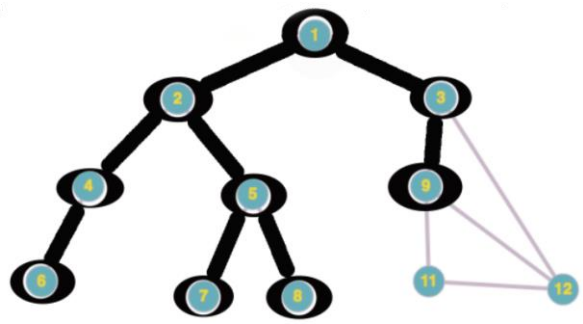
У вершины **6** нет больше смежных вершин, кроме тех, которые мы уже проходили. Тогда возвращаемся к **4**. Видим, что у нее тоже смежных нет. Рассматриваем **2**. От нее идем по ребру к **5**.



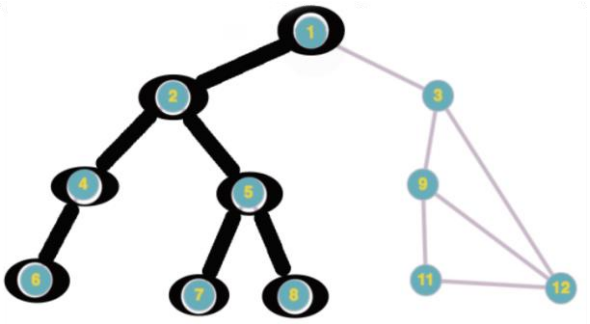
Встречаем **7**.



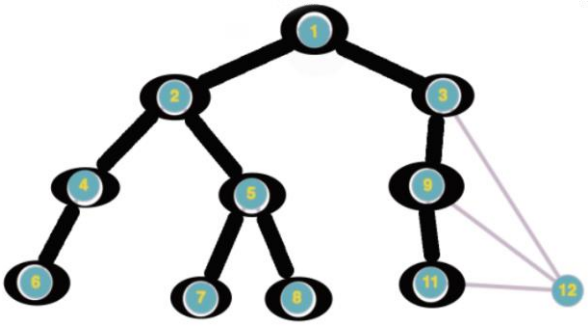
Далее к **9**.



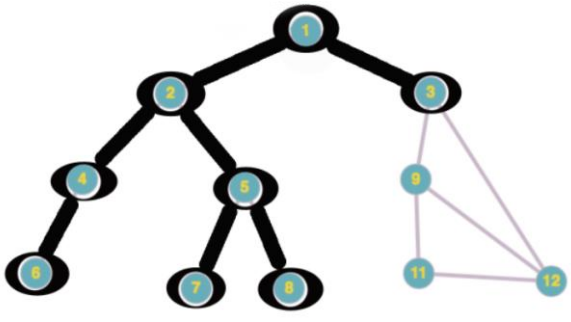
Возвращаемся к вершине **5**, от нее спускаемся к **8**.



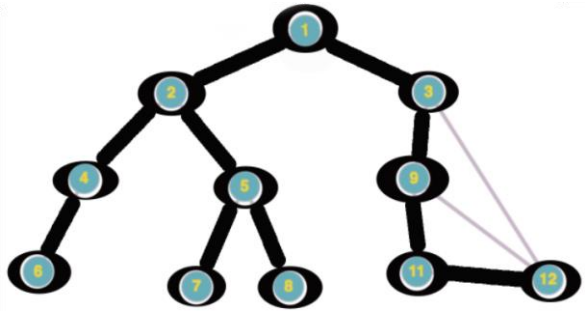
От **9** находим **11**.



Поднимаемся наверх, через **5** и **2**, оказываемся у **1**, от нее двигаемся к непомеченным вершинам. Это **3**.



И, наконец, от **11** у нас есть ребро, ведущее к **12**. Конец алгоритма.



При данном методе можно использовать цвета вершин.

Каждая непомеченная вершина изначально имеет белый цвет. Когда вершину обнаруживают в первый раз, ее цвет меняется на серый. Вершина является обработанной тогда и только тогда, когда просмотрены все ее смежные вершины. Когда вершина обработана, она меняет цвет на черный.

Каждой вершине *v* будем устанавливать две метки времени:
$In(v)$ – время, когда вершина обнаружена и ее цвет изменен на серый.
$Out(v)$ – время, когда вершина обработана и цвет изменен на черный.

Эти метки в дальнейшем будут использоваться в алгоритмах для поиска и выявления компонент сильной связности.

Каждая вершина *v* соответствует неравенству $In(v) < Out(v)$.
Вершина v будет иметь белый цвет до момента времени $In(v)$, серый цвет с $In(v)$ до $Out(v)$ и станет черной после $Out(v)$.

Поиск в глубину используется для классификации ребер графа.
Существуют 4 типа ребер:

– *Дуги дерева* - ребра, которые ведут к вершинам и раньше не посещались. Они осуществляют функцию формирования глубинного остовного леса для заданного графа.

– *Обратные дуги* -дуги, которые в остовном лесу идут от потомков к предкам. Дуга, которая идет из вершины в саму себя, считается обратной.

– *Прямые дуги*- дуги, которые идут от предков к собственным потомкам, но, тем не менее, не являются дугами дерева.

– *Поперечные дуги*- ребра, которые соединяют вершины, которые не являются предками или потомками.

Ребро (u, v) является:

а) *ребром дерева* или *прямым ребром* тогда и только тогда, когда выполняется неравенство

$$In(u) < In(v) < Out(v) < Out(u)$$

б) *обратным ребром* тогда и только тогда, когда выполняется неравенство

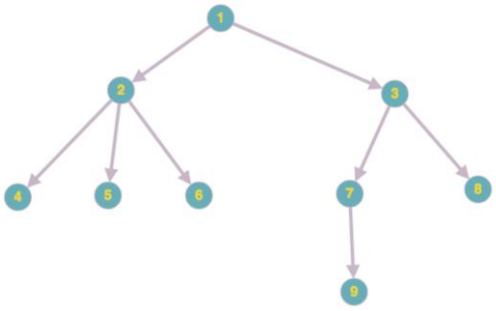
$$In(v) < In(u) < Out(u) < Out(v)$$

в) *поперечным ребром* тогда и только тогда, когда выполняется неравенство

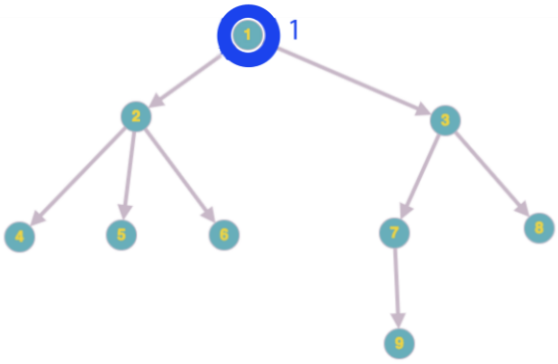
$$In(v) < Out(v) < In(u) < Out(u).$$

Алгоритм работает за время $O(|V| + |E|)$.

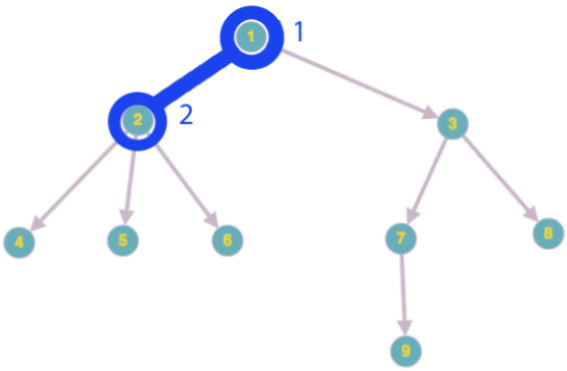
Рассмотрим поиск DFS с отметками времени.



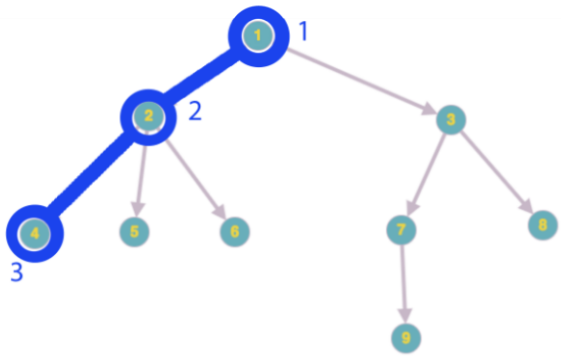
Вершину **1** начинаем обрабатывать в момент времени «1».



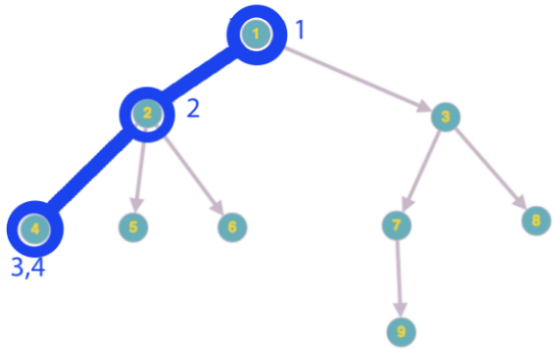
Из вершины **1** мы пошли в вершину **2**. Записываем время «2».



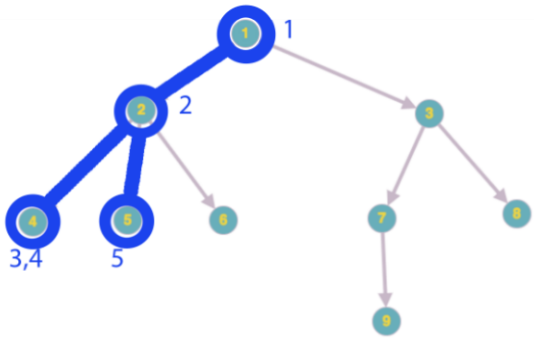
Далее спускаемся к **4**. Время ее начала обработки «3».



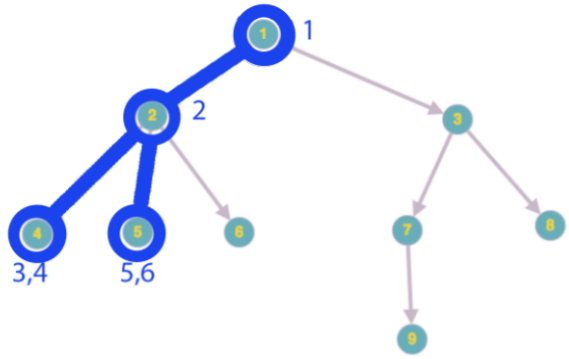
Из вершины **3** идти некуда, мы закончили ее обработку. Время окончания «4».



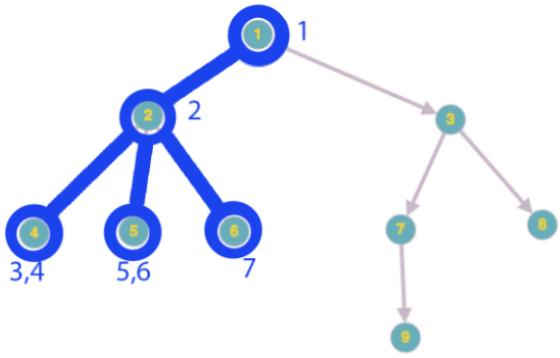
Вернувшись к **2**, далее спускаемся по ребру к **5**. Время «5».



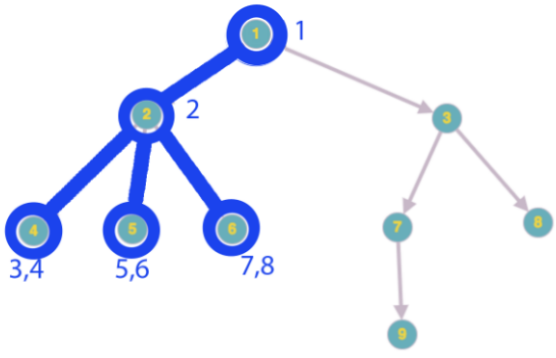
Из **5** идти тоже больше некуда, записываем время конца обработки «6».



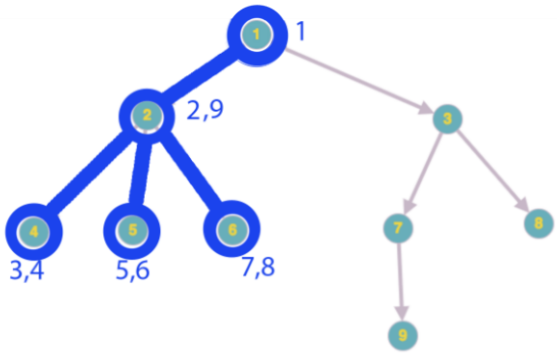
Так же с **6**.



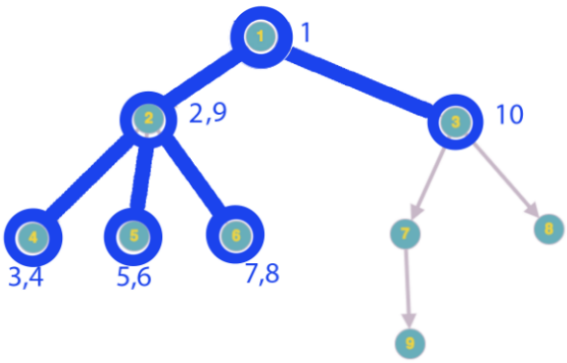
Время окончания «8».



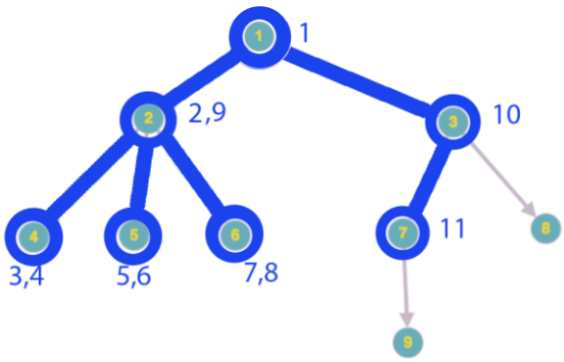
Полностью обработав вершину **6**, и возвратившись к **2**, видим, что все смежные с ней вершины мы прошли. Тогда время конца обработки вершины **2** – это «9».



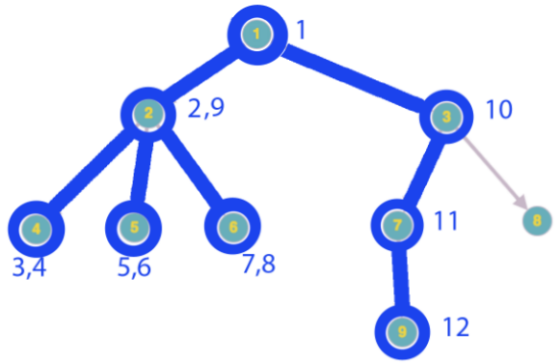
Поднимаемся к **1**, от нее двигаемся вглубь по непройденным вершинам. Вершине **3** приписываем время начала обработки «10».



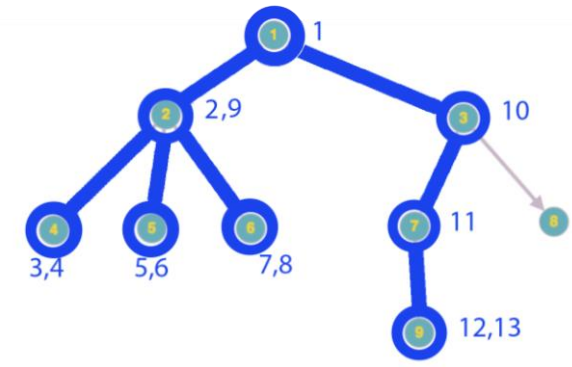
У **7** время начала равно «11».



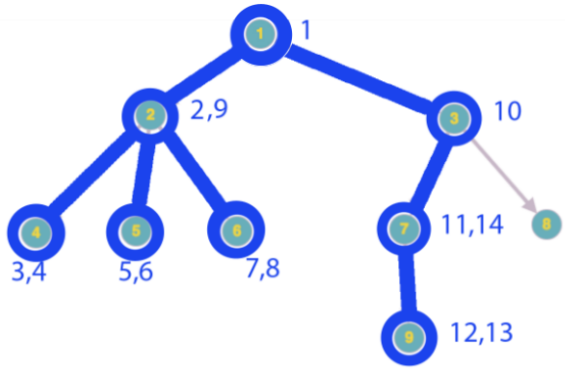
Двигаясь вглубь графа, встречаем **9**. Приписываем «12».



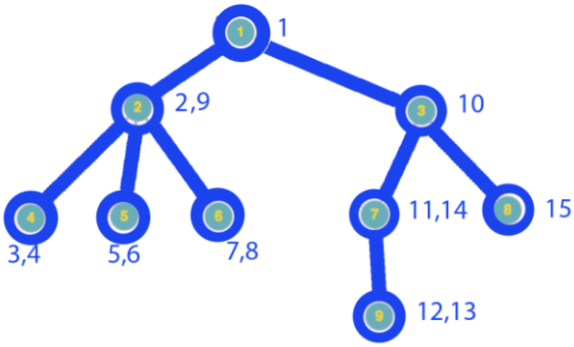
От **9** нам двигаться больше некуда, мы заканчиваем ее обработку и приписываем время «13».



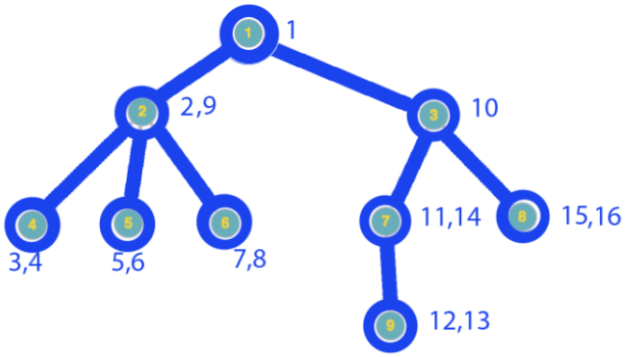
Аналогично к **7** приписываем время окончания «14».



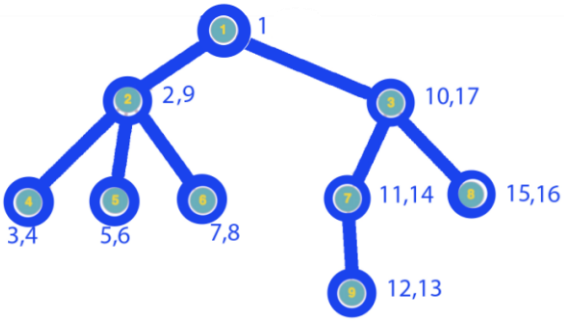
У **10** осталась одна смежная вершина – это вершина **8**. Записываем время начала обработки «15».



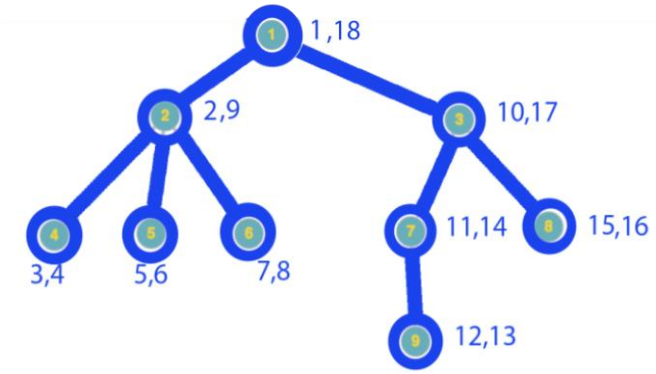
И время конца обработки «16».



Поднимаемся выше, у **3** все смежные вершины помечены. Приписываем «17».

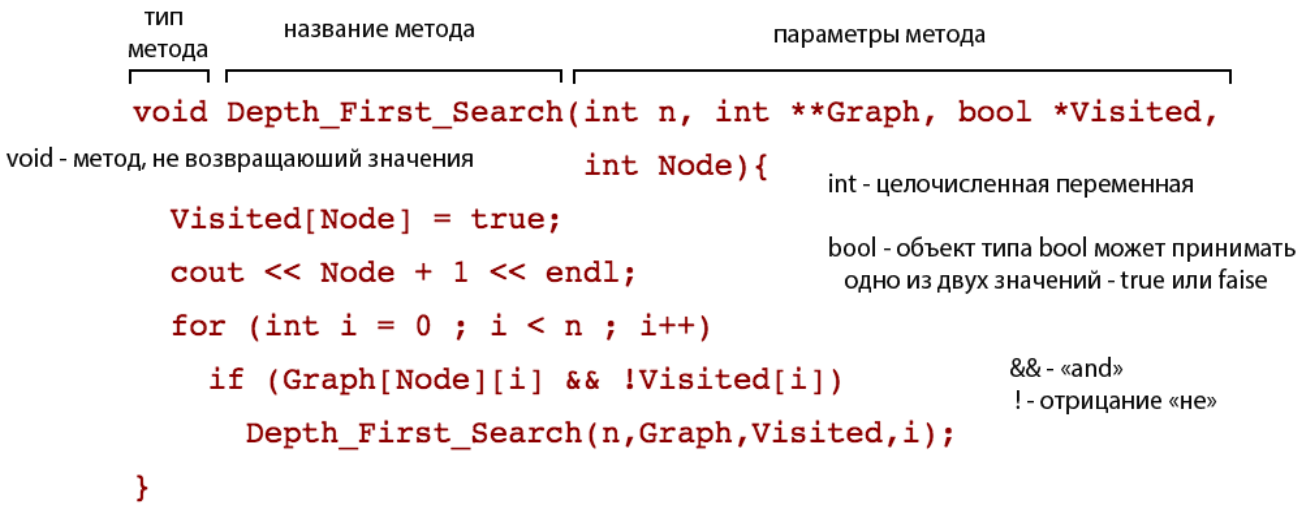


Наконец, возвращаемся к **1**. Время конца обработки «18».



Конец поиска.

Описание функции алгоритма DFS поиска в глубину



Применение алгоритма

1) Поиск любого пути в графе,
2) Поиск лексикографически первого пути в графе,
3) Проверка, является ли одна вершина дерева предком другой,
4) Задача LCA – наименьший общий предок,
5) Топологическая сортировка,
6) Проверка графа на ацикличность и нахождение цикла,
7) Поиск компонент сильной связности,
8) Поиск мостов.

Компоненты сильной связности SCC

Компонента сильной связности SCC (Strongly Connected Component) – максимальное подмножество вершин V таких, что любые две достижимы друг из друга, то есть $\forall\, v, u \in V \quad v \mapsto u, u \mapsto v$.

Сопутствующие определения:

Вершины *v* и *u* называются *сильно связными*, если существует путь от *v* к *u*, от *u* к *v*.

Орграф называется *сильно-связным*, если любые две вершины сильно связны.

Любая вершина сильно связна с собой.

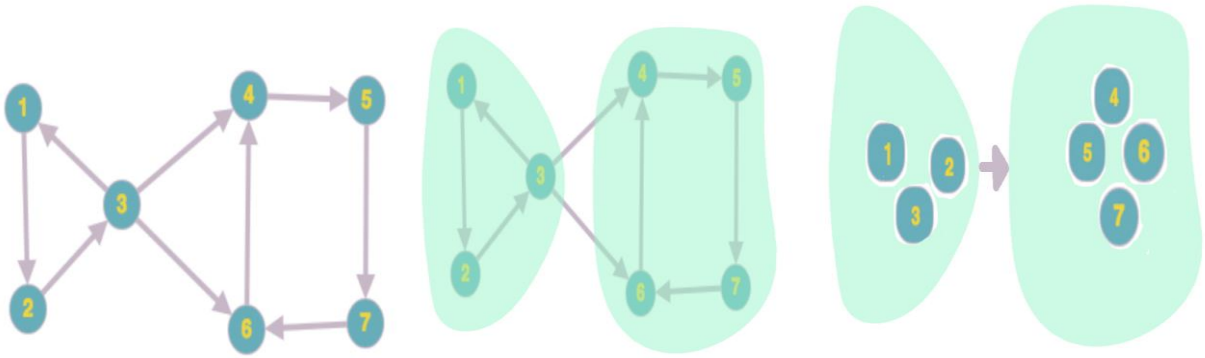
Метаграф (конденация графа) – граф, полученный сжатием каждой компоненты связности в одну вершину.

Метаграф является ацикличным.

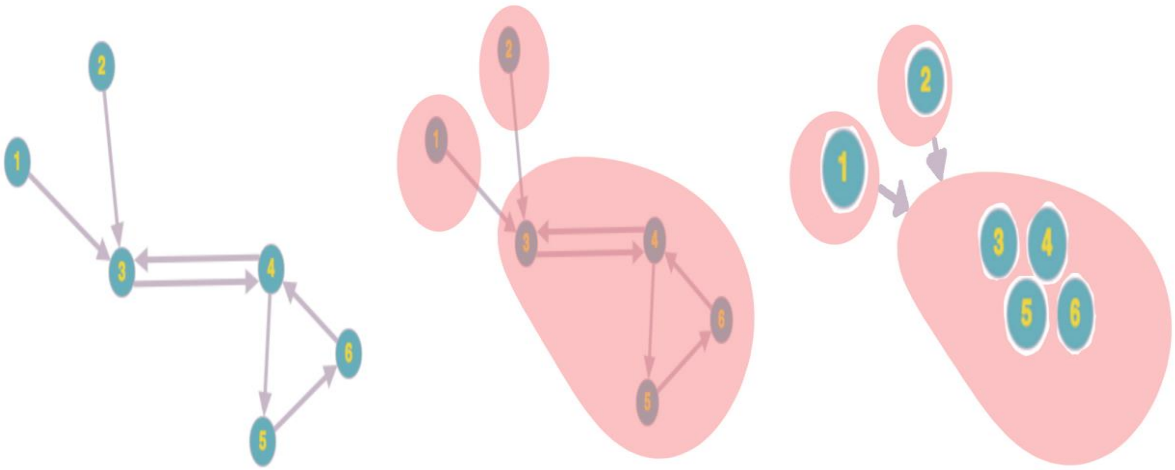
DAG – directed acyclic graph – ориентированный ациклический граф.

Примеры SCC в орграфах

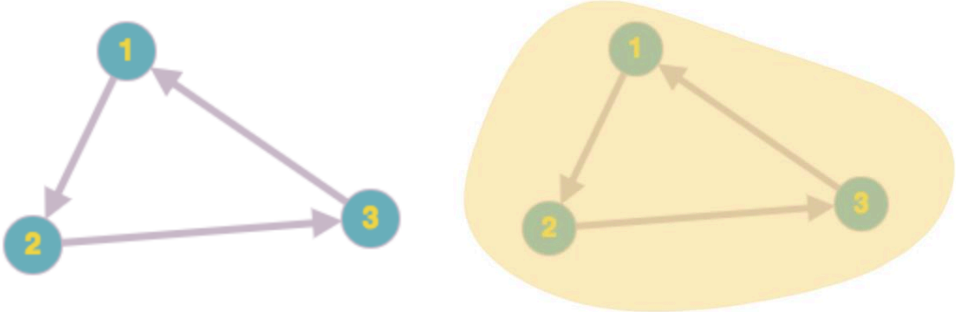
Пример 1 - граф, граф с компонентами SCC и метаграф



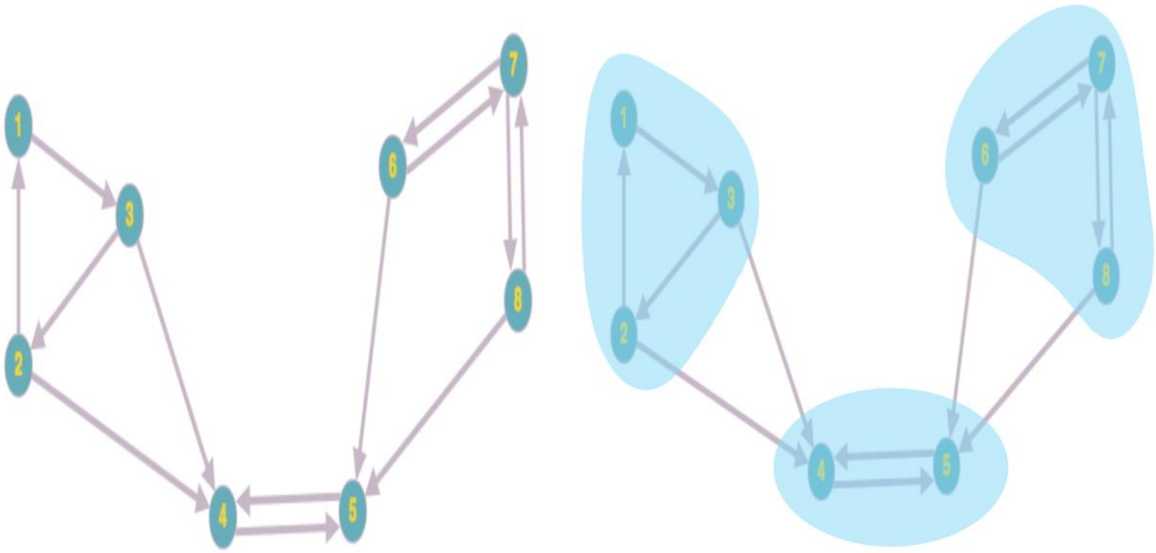
Пример 2



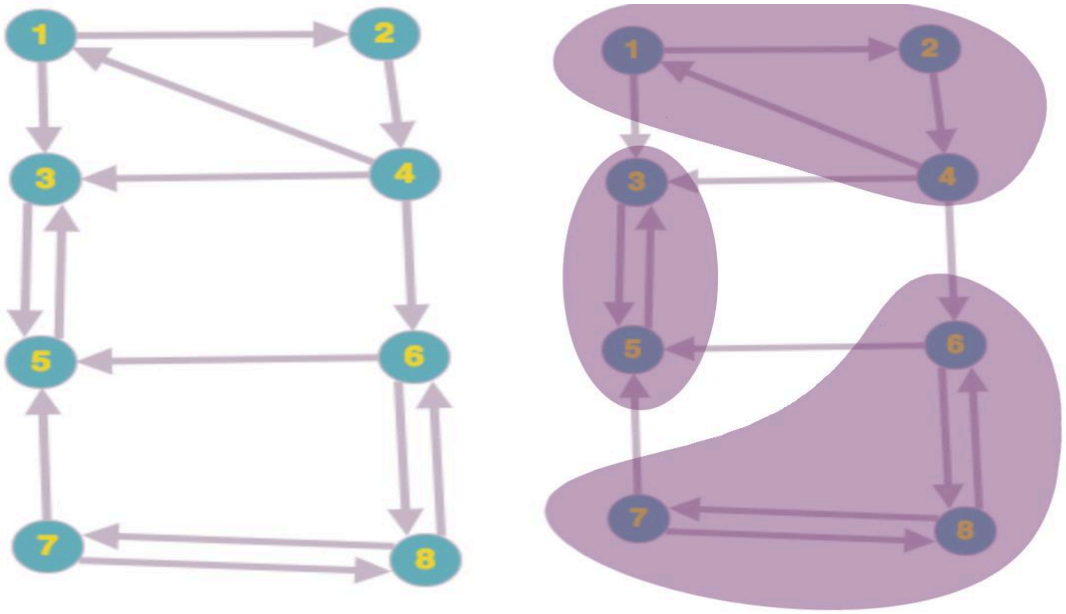
Пример 3



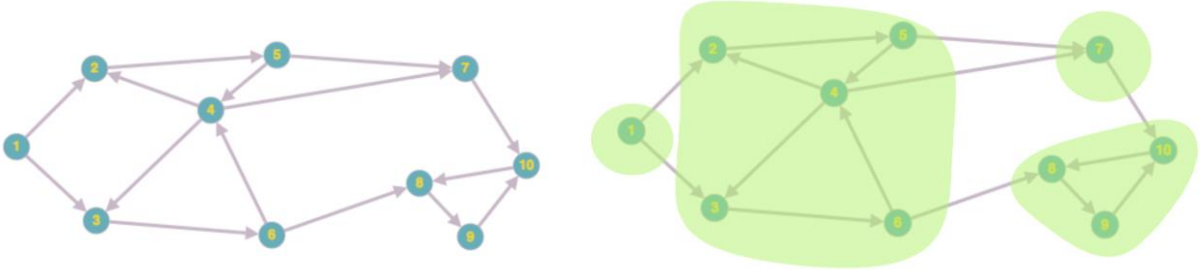
Пример 4



Пример 5

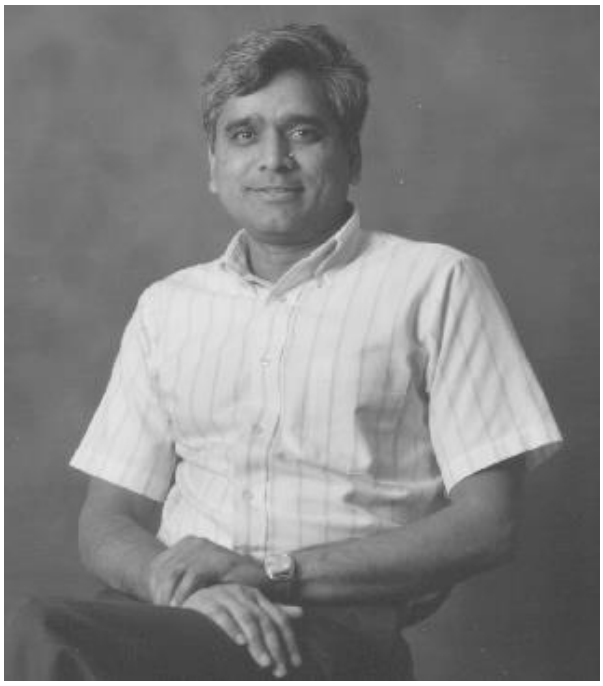


Пример 6



Алгоритм Косарайю

Самбасив Рао Косарайю – американский ученый с индийскими корнями, профессор информатики в Университете Хопкинса. В 1978 написал работу, в которой описал метод эффективного нахождения компонент SCC. Позднее данный метод стал известен как «Алгоритм Косарайю». В 1983 году алгоритм был опубликован в книге А. Ахо, Д. Хопкрофта и Д. Ульмана «Структуры данных и алгоритмы» («Data Structures and Algoritms»).



Данный алгоритм выявляет компоненты SCC графа $G = (V, E)$ за время $O(|V| + |E|)$.

Его сложность связана со сложностью алгоритма поиска DFS, который должен быть запущен дважды, а также со сложностью нахождения обратного графа. Оба эти алгоритма имеют линейную сложность, и поэтому Алгоритм Косарайю работает за линейное время.

Для начала выполняется DFS-поиск для обратного графа (графа, полученного при инвертировании ребер исходного).

Далее идет вычисление вектора обратного порядка, который присваивает вершинам индексы в порядке, в котором заканчивается обработка вершин в DFS.

На втором шаге происходит поиск DFS на исходном графе, причем вершины берутся в том порядке, который является обратным вектору обратного порядка обхода при первом запуске.

Когда выполняется метод поиска DFS, то используют непосещенные вершины, которые имеют максимальный номер.

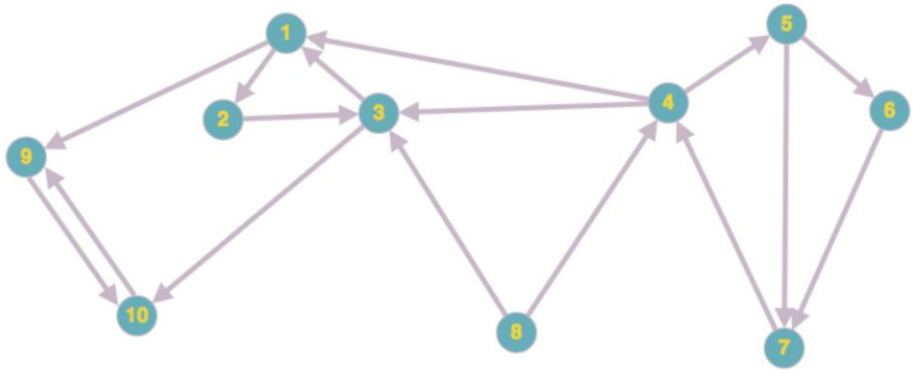
После завершения алгоритма получается лес, у которого деревьями будут являться компонеты SCC.

В сокращенном варианте весь алгоритм Косарайю можно записать так:

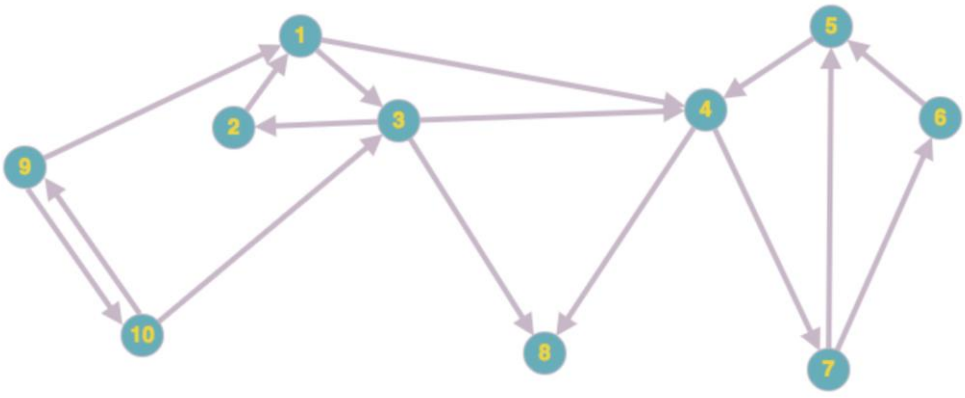
1) Строим *G*-reversed,
2) DFS (*G-r*) – запуск DFS на *G-r,*
3) DFS (*G*), перебирая вершины в порядке убывания post-значений, найденный на шаге 2.

Пример Алгоритма Косарайю

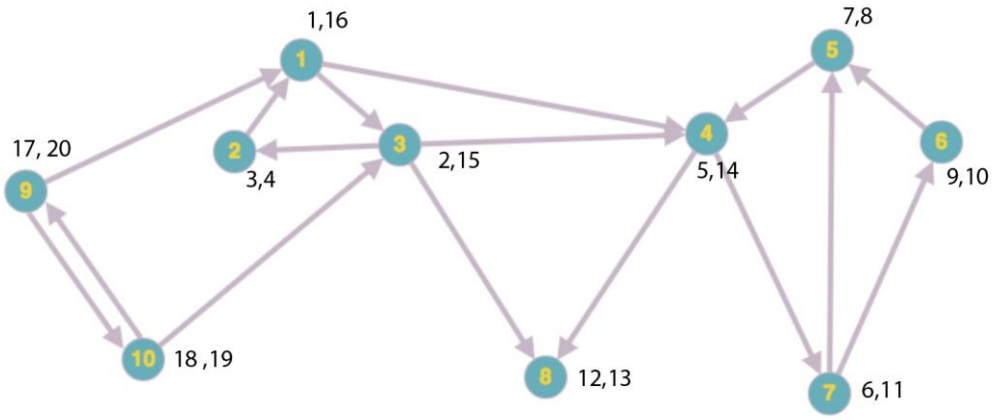
Граф *G*



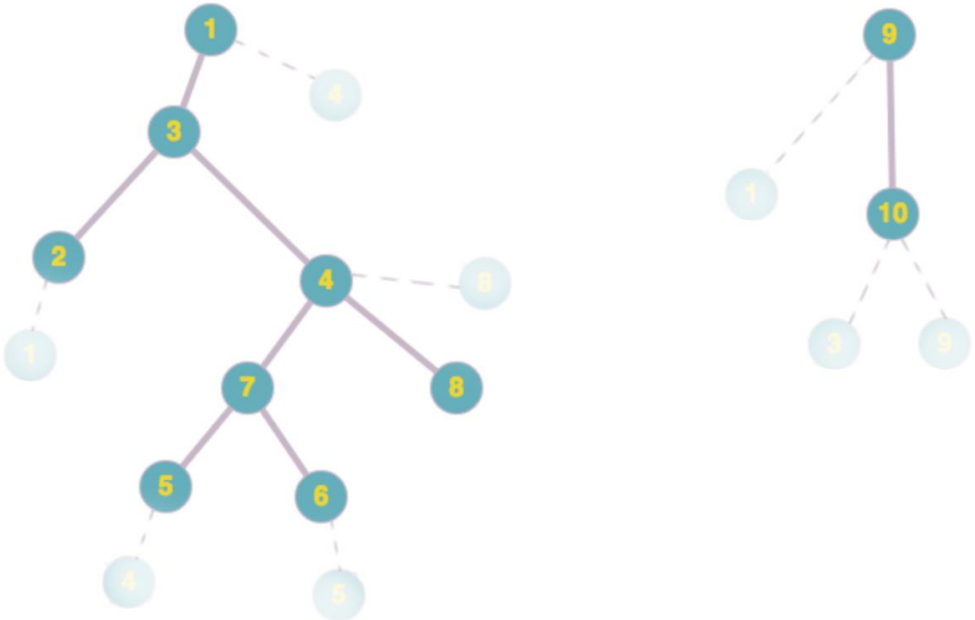
G-reversed (далее *G-r*)



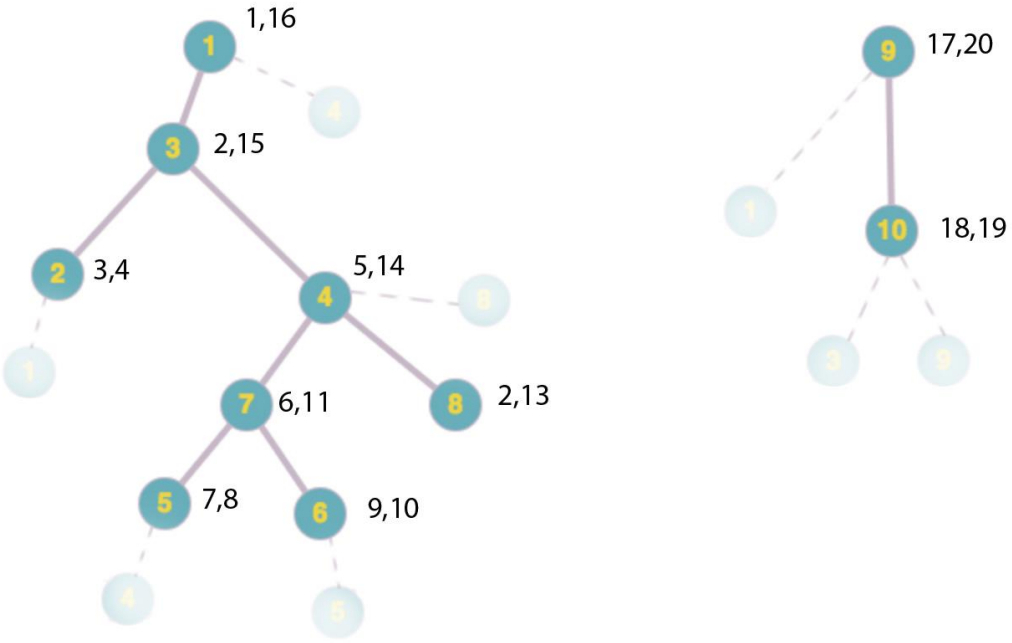
Запуск DFS (*G-r*)



Дерево поиска DFS



Дерево поиска DFS с отметками времени обработки



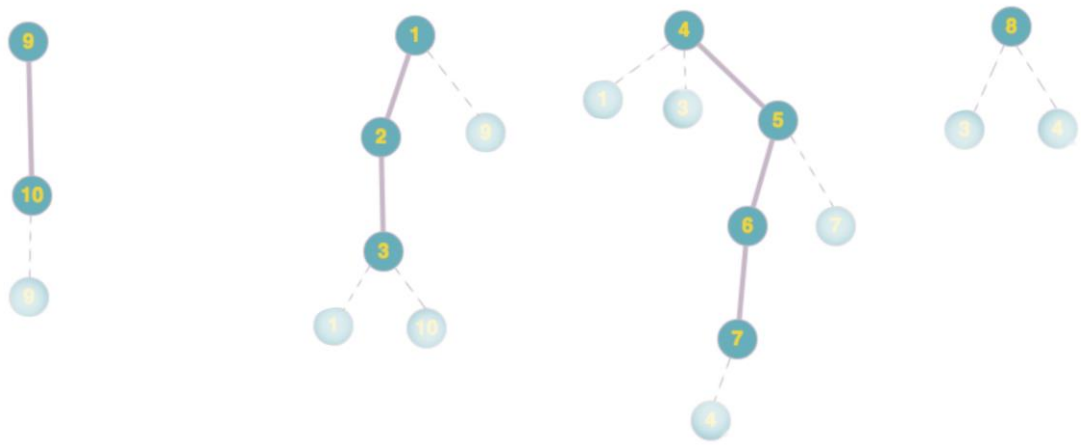
Полученный вектор обратного порядка

У вершины **9** время окончания обработки максимальное («20»), поэтому в векторе у нее индекс «9»

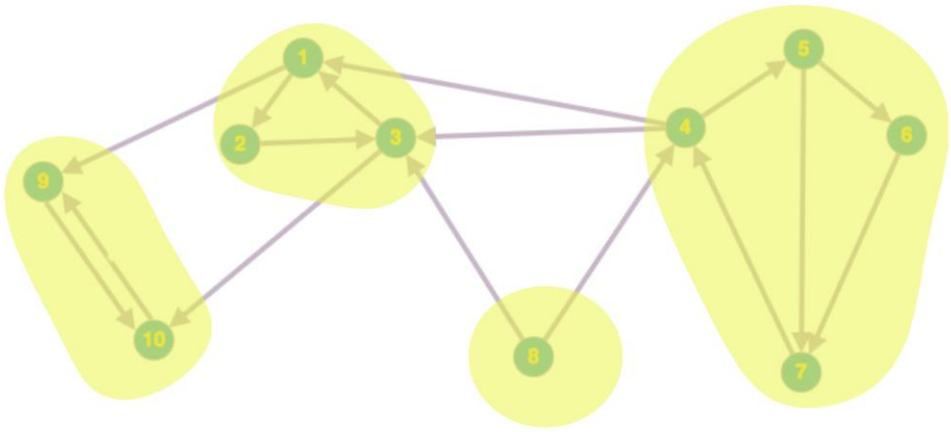
0	1	2	3	4	5	6	7	8	9
2	5	6	7	8	4	3	1	10	9

Деревья, полученные в результате второго запуска DFS на исходном графе (вершины берутся в порядке убывания их индексов в векторе).

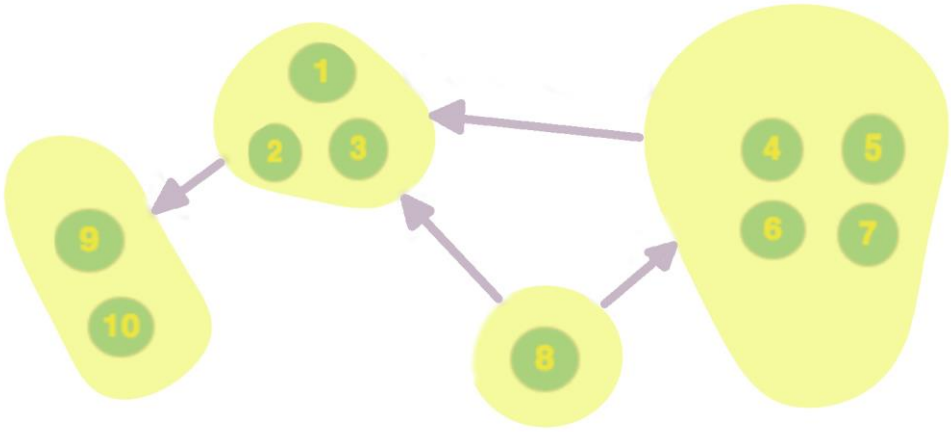
Деревья являются компонентами SCC на графе *G* (как, впрочем, и на графе *G-r*)



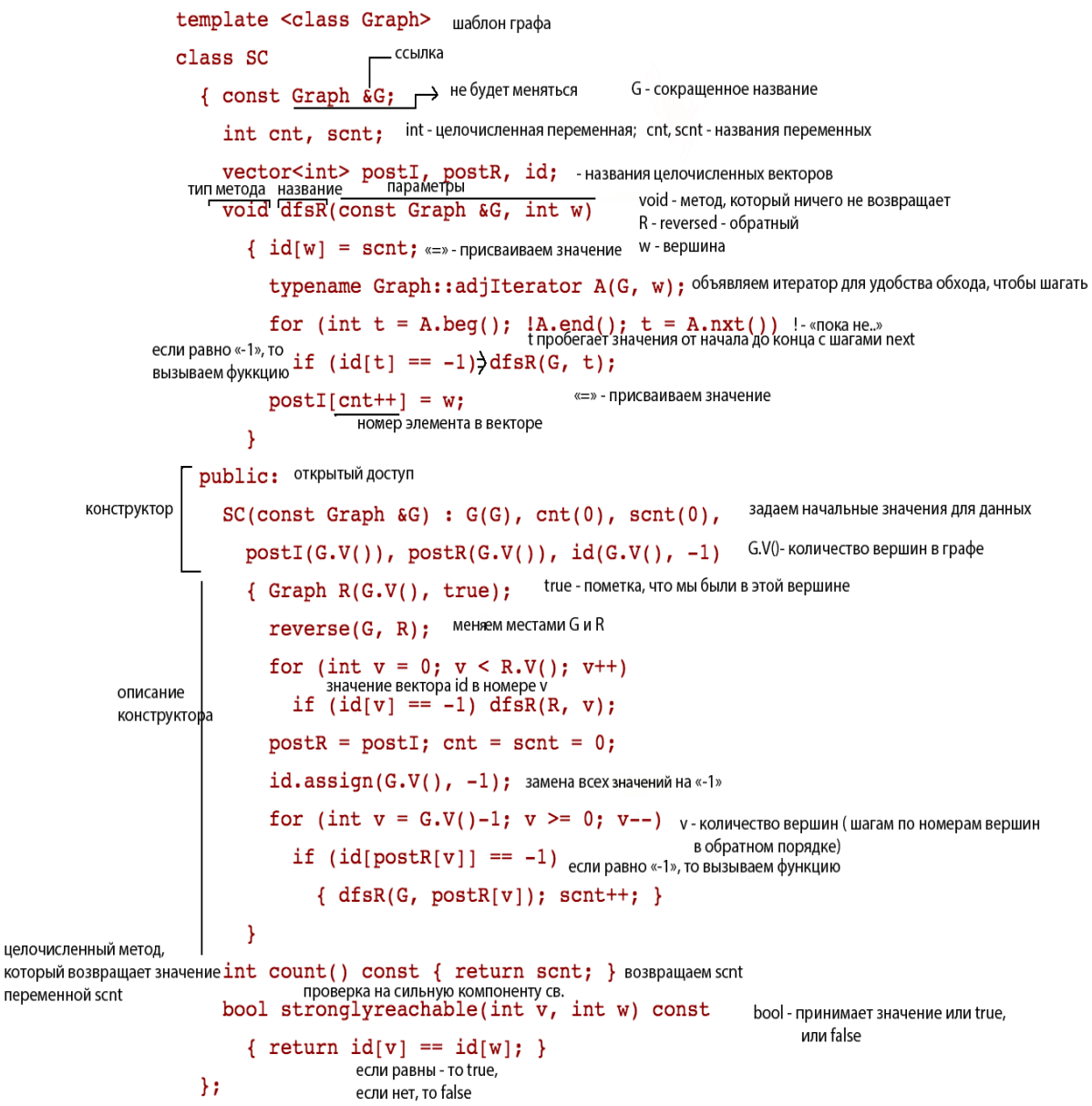
Компоненты SCC



Метаграф *G*



Описание Алгоритма Косарайю



Реализация Алгоритма с возможностью ввода данных графа

Если предыдущий алгоритм следовал из описания статей, то этот алгоритм написан заново и проверен (см. ниже).

```
#include <iostream>
#include <vector>

const int dim = 100;

//граф и транспонированный граф
std::vector < std::vector<int> > G (dim, std::vector<int>(dim)),
Gt(dim, std::vector<int>(dim));
//метка узла
std::vector<char> used;
//упор€доченный список и список SCC (Strongly Connected Components)
std::vector<int> order, SCC;

//обход в глубину (depth first search) по графу и топологическа€
сортировка
//список order с вершинами в пор€дке увеличени€ времени выхода
void dfs1(int v)
{
        used[v] = true;
        for (size_t i=0; i<G[v].size(); i++)
                if (!used[ G[v][i] ])
                        dfs1 (G[v][i]);
        order.push_back (v);
}

//обход в глубину по транспонированному графу и поиск компонент
сильной св€зности
void dfs2(int v)
{
        used[v] = true;
        SCC.push_back (v);
        for (size_t i=0; i<Gt[v].size(); i++)
                if (!used[ Gt[v][i] ])
                        dfs2 (Gt[v][i]);
}

int main()
{
        //количество узлов и р€бер графа
        int N, M;
        std::cout << "Number of vertices, N = ";
        std::cin >> N;
        std::cout << "Number of edges, M = ";
        std::cin >> M;
        std::cin.sync();

        //р€бра
        int a,b;
        std::cout << "\nEnter edges";
        for (int i = 0; i<M; i++)
        {
                std::cout << "\n\tFrom vertex ";
                std::cin >> a;
                std::cin.sync();
```

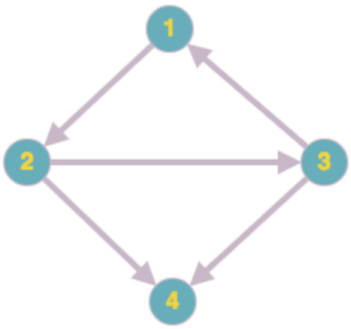
```
                std::cout << "\tTo vertex ";
                std::cin >> b;
                std::cin.sync();
                G[a-1].push_back(b-1);
                Gt[b-1].push_back(a-1);
        }

        std::cout << "\nSCC list";
        used.assign (N, false);
        for (int i=0; i<N; ++i)
                if (!used[i])
                        dfs1 (i);

        used.assign (N, false);
        for (int i=0; i<N; ++i)
        {
                int v = order[N-1-i];
                if (!used[v]) {
                        dfs2 (v);
                        //вывод компоненты SCC
                        std::cout << "\n";
                        for (size_t i=0; i<SCC.size(); i++)
                                std::cout << "\t" << SCC[i]+1;
                        SCC.clear();
                }
        }

        getchar();
        return 0;
}
```

Примеры работы программы

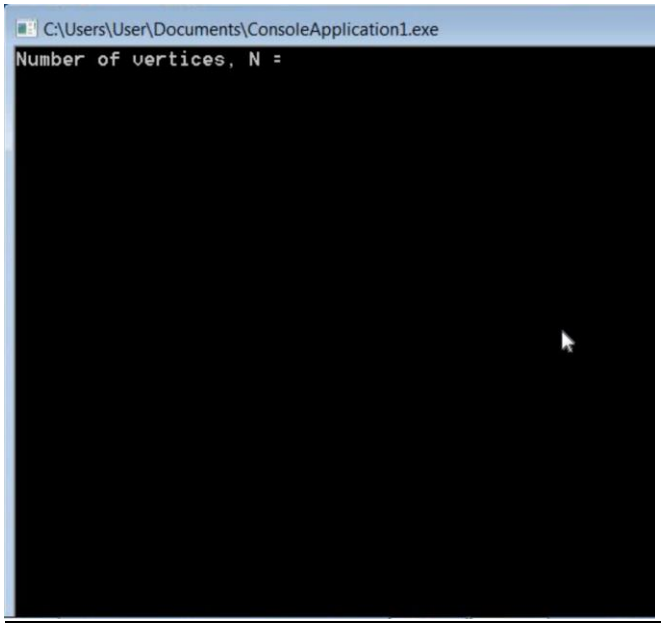
(Скриншоты)

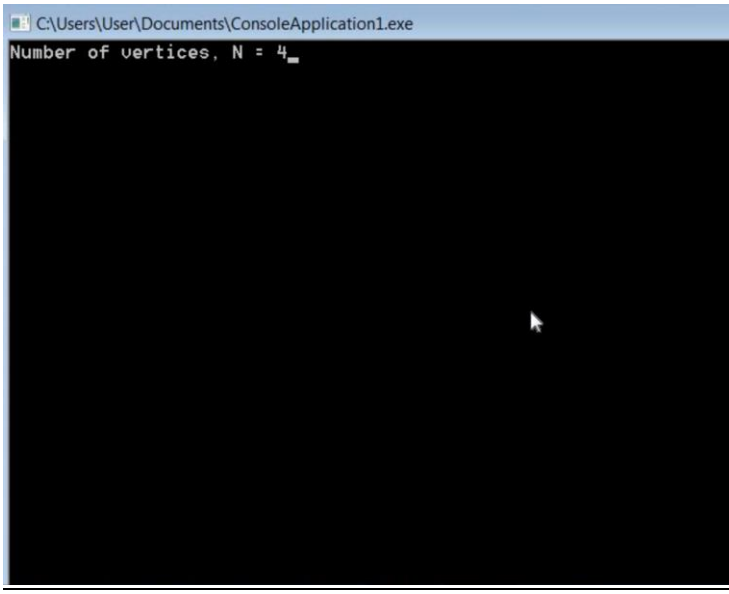
1)

Заданный граф *G*

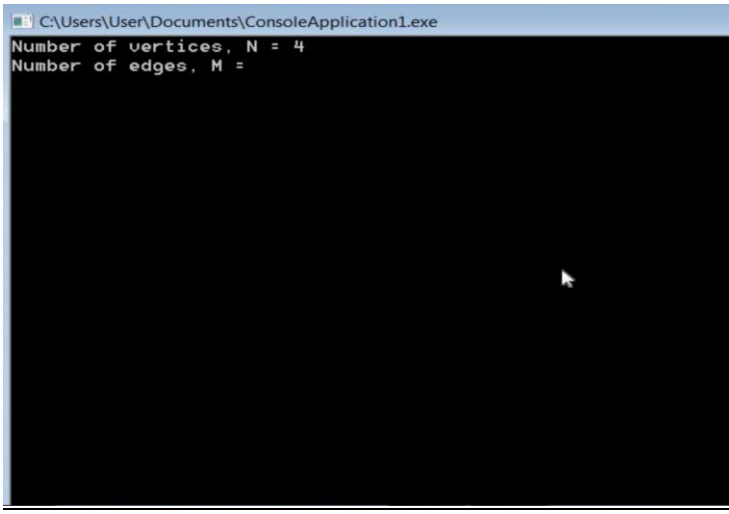


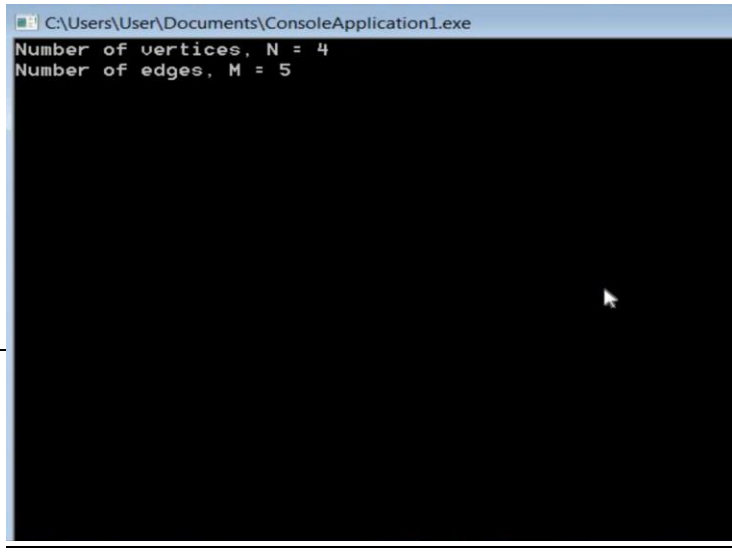
Ввод N – количества вершин



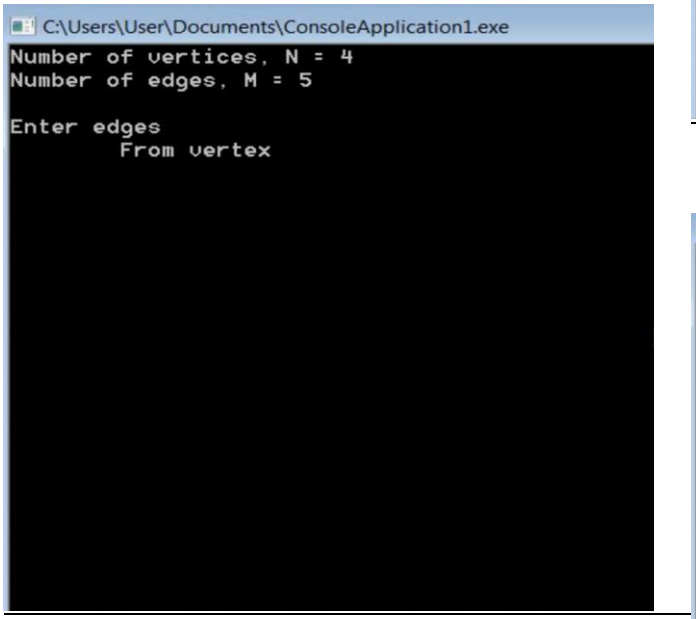


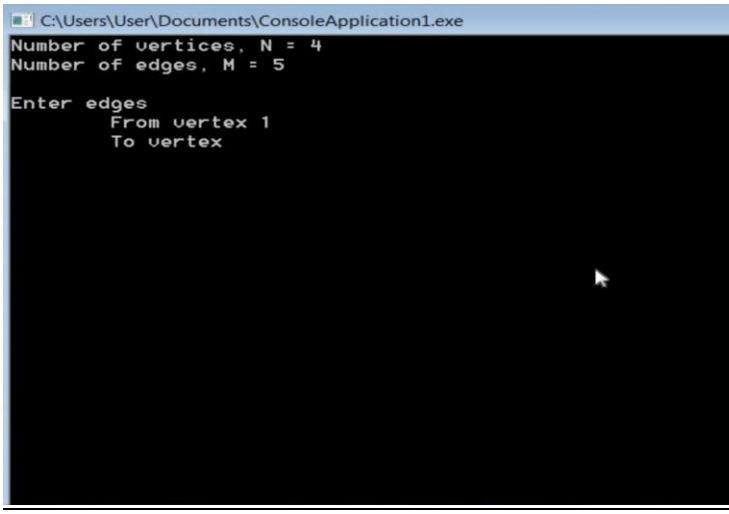
Ввод M – количества ребер

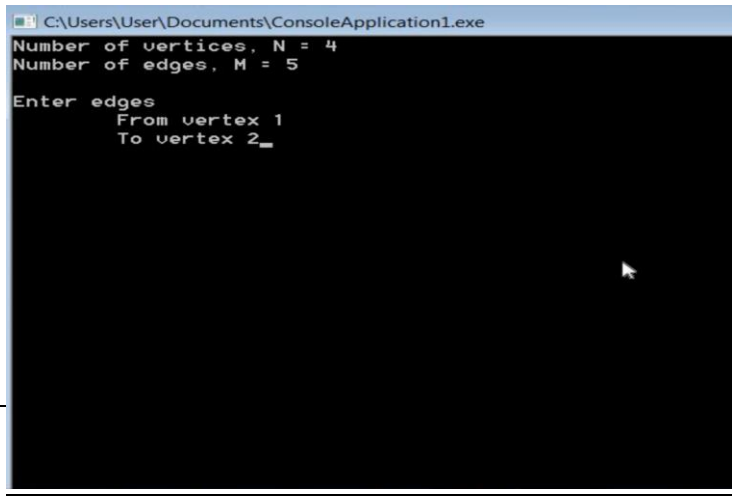


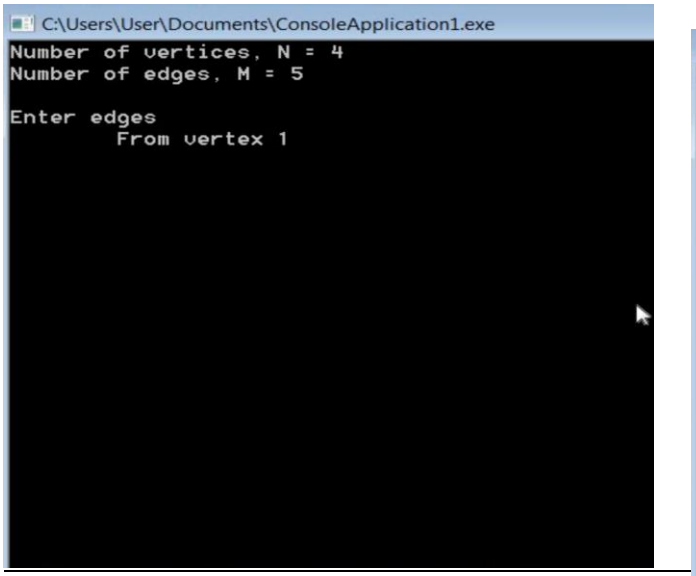


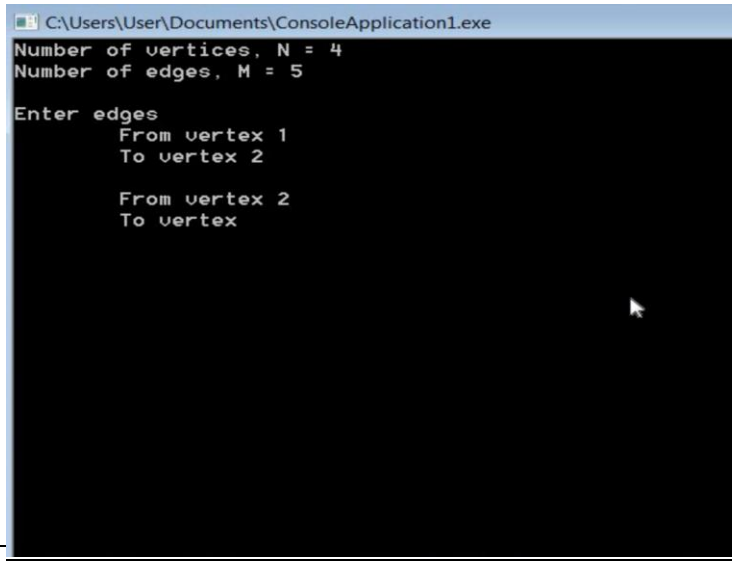
Ввод данных о каждом существующем ребре (from – to) – требуется ввести начало и конец каждой дуги



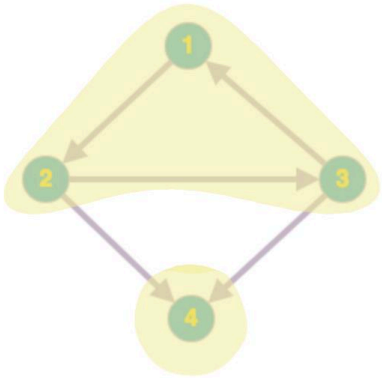








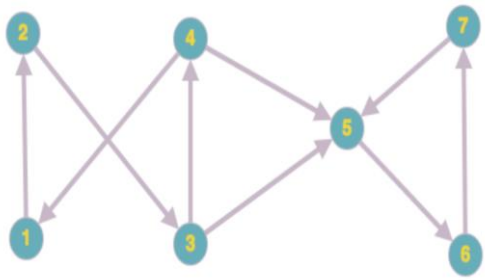
```
C:\Users\User\Documents\ConsoleApplication1.exe
Number of vertices, N = 4
Number of edges, M = 5

Enter edges
        From vertex 1
        To vertex 2

        From vertex 2
        To vertex 3

        From vertex 3
        To vertex 1

        From vertex 2
        To vertex 4

        From vertex 3
        To vertex 4
```

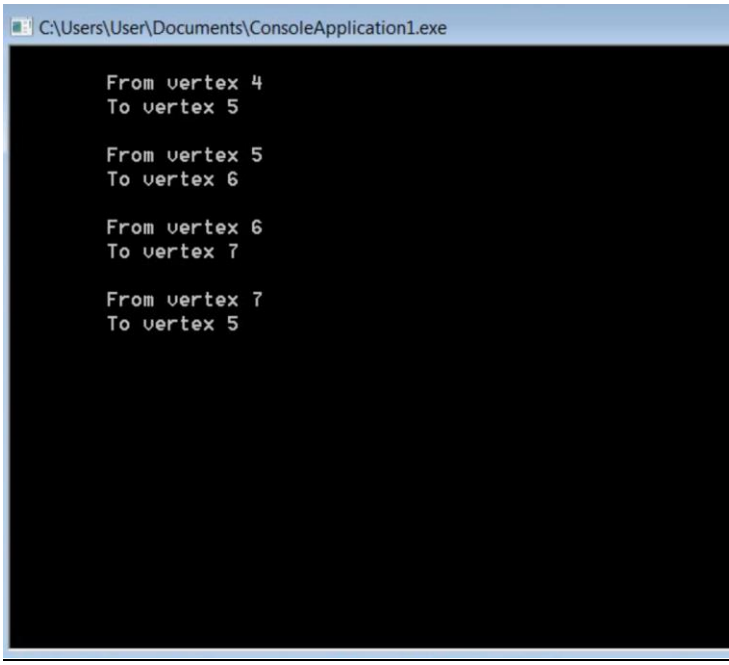


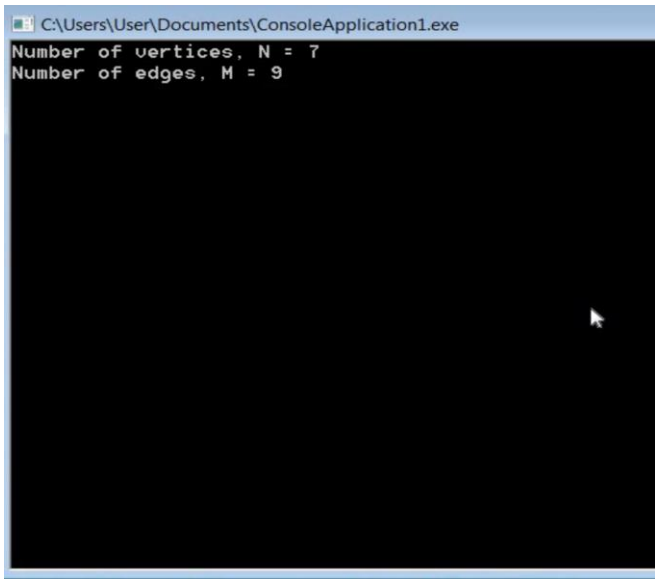
Вывод компонент SCC

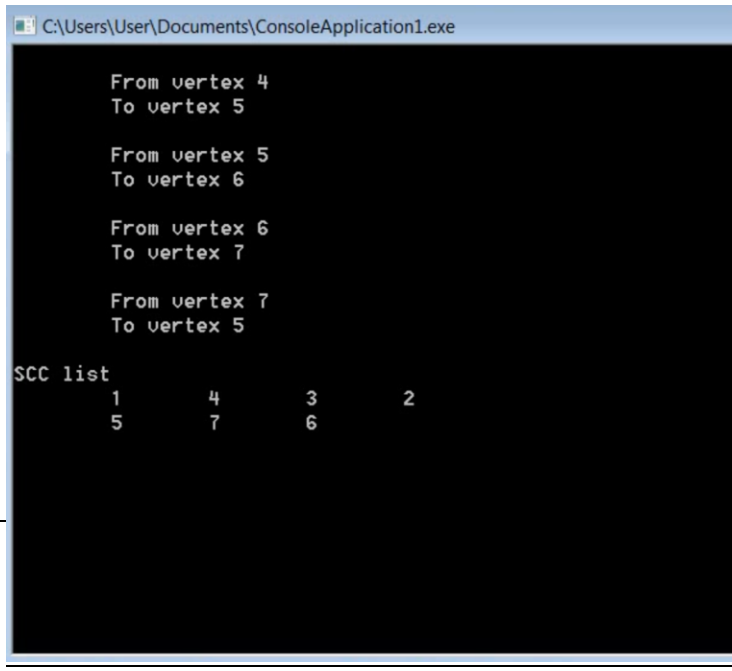
```
C:\Users\User\Documents\ConsoleApplication1.exe
Number of vertices, N = 4
Number of edges, M = 5

Enter edges
        From vertex 1
        To vertex 2

        From vertex 2
        To vertex 3

        From vertex 3
        To vertex 1

        From vertex 2
        To vertex 4

        From vertex 3
        To vertex 4

SCC list
        1       3       2
        4
```

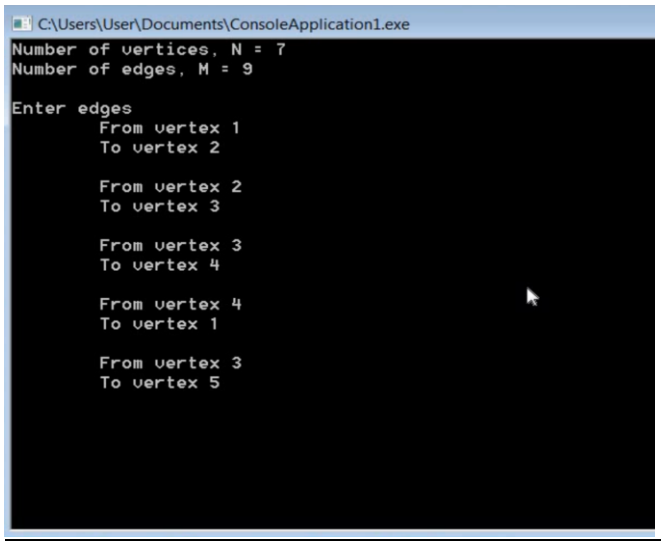
2)

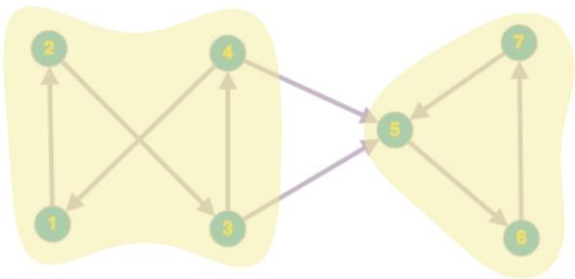

2
4
7
5
1
3
6


C:\Users\User\Documents\ConsoleApplication1.exe
From vertex 4
To vertex 5
From vertex 5
To vertex 6
From vertex 6
To vertex 7
From vertex 7
To vertex 5

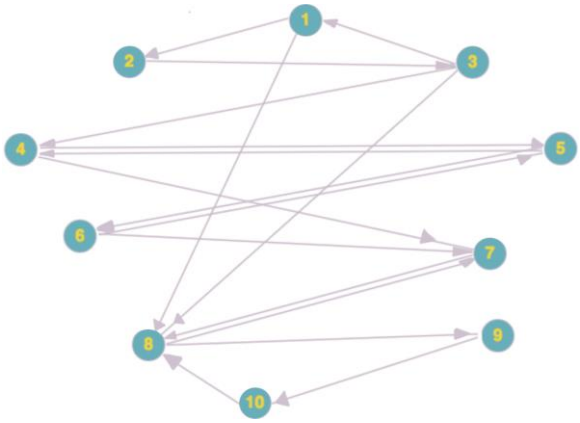

C:\Users\User\Documents\ConsoleApplication1.exe
Number of vertices, N = 7
Number of edges, M = 9


C:\Users\User\Documents\ConsoleApplication1.exe
From vertex 4
To vertex 5
From vertex 5
To vertex 6
From vertex 6
To vertex 7
From vertex 7
To vertex 5
SCC list
1 4 3 2
5 7 6

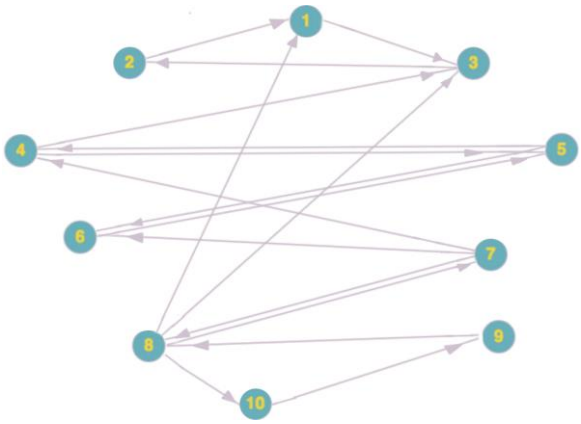

C:\Users\User\Documents\ConsoleApplication1.exe
Number of vertices, N = 7
Number of edges, M = 9
Enter edges
From vertex 1
To vertex 2
From vertex 2
To vertex 3
From vertex 3
To vertex 4
From vertex 4
To vertex 1
From vertex 3
To vertex 5


2
4
7
5
1
3
6

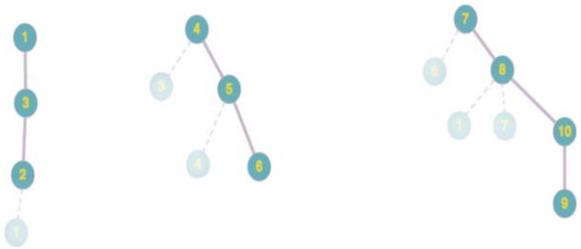
3) Граф *G*



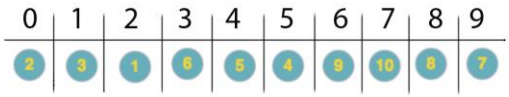
G-reversed



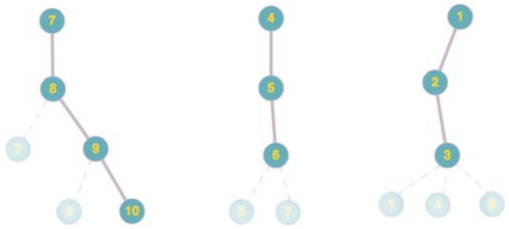
Деревья, полученные при первом запуске DFS



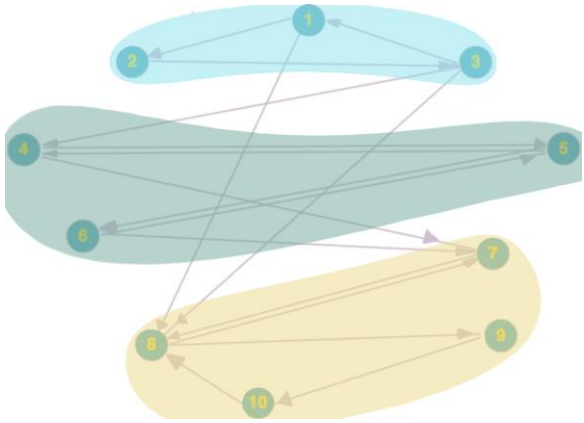
Вектор обратного порядка обхода

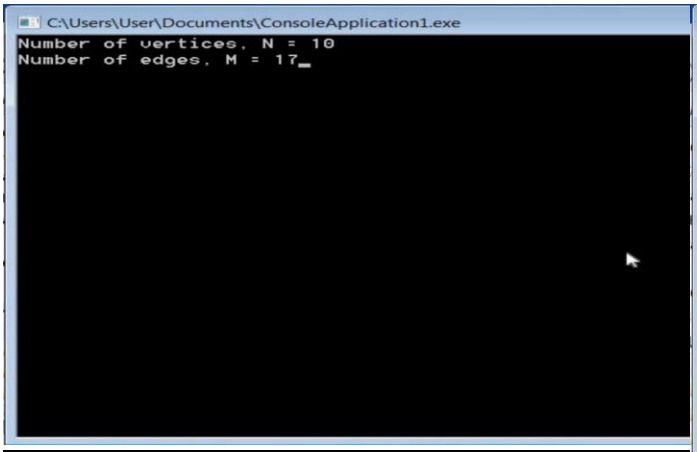


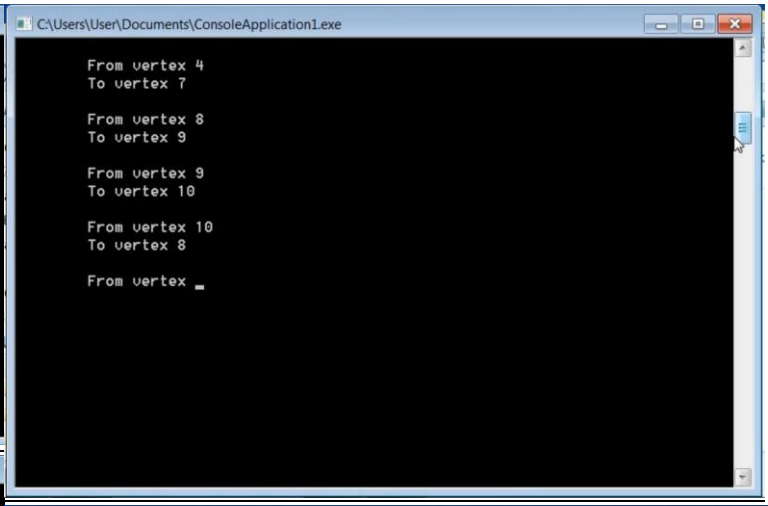
Полученные компоненты SCC

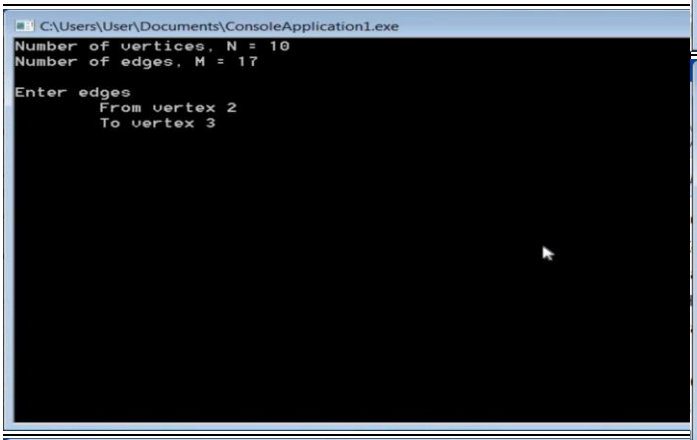


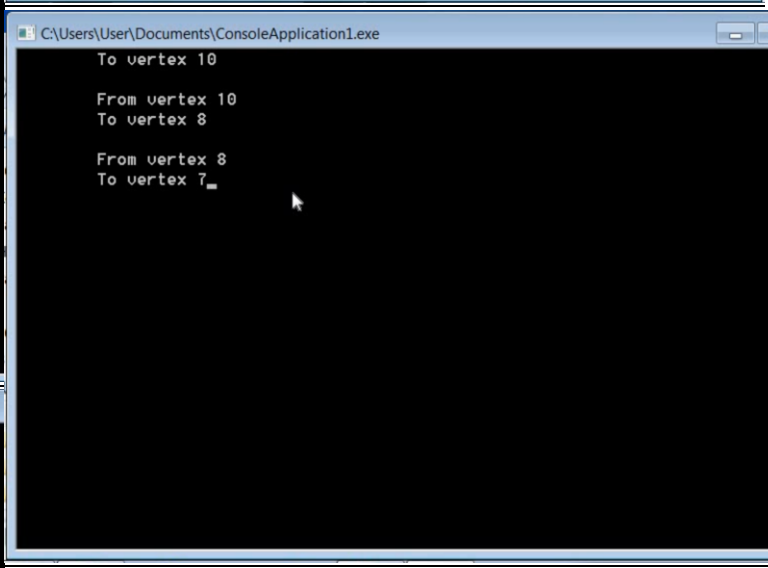
SCC

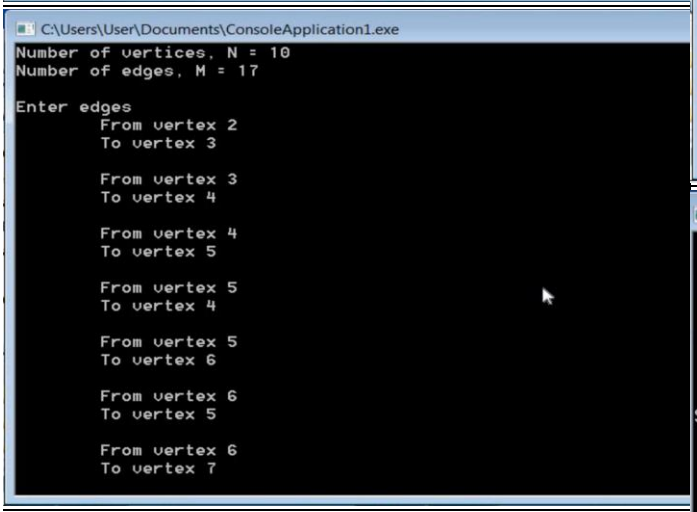


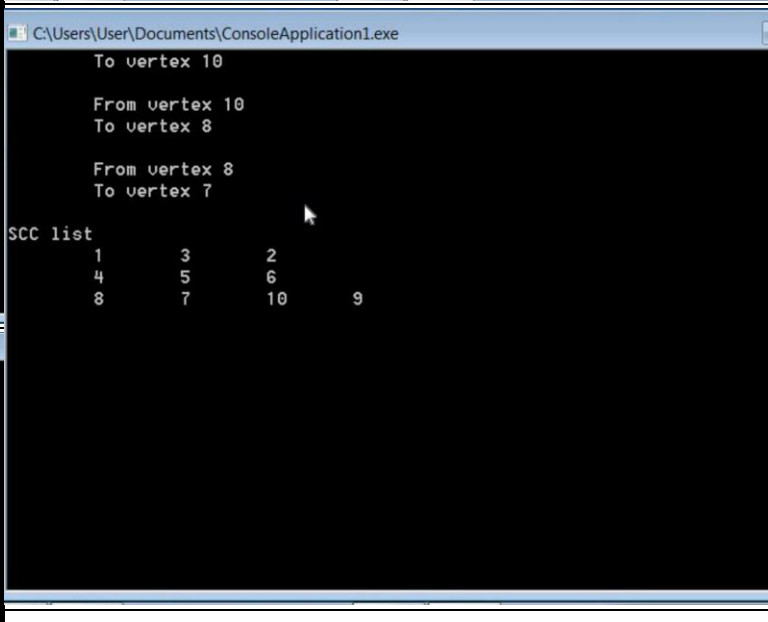

C:\Users\User\Documents\ConsoleApplication1.exe
Number of vertices, N = 10
Number of edges, M = 17

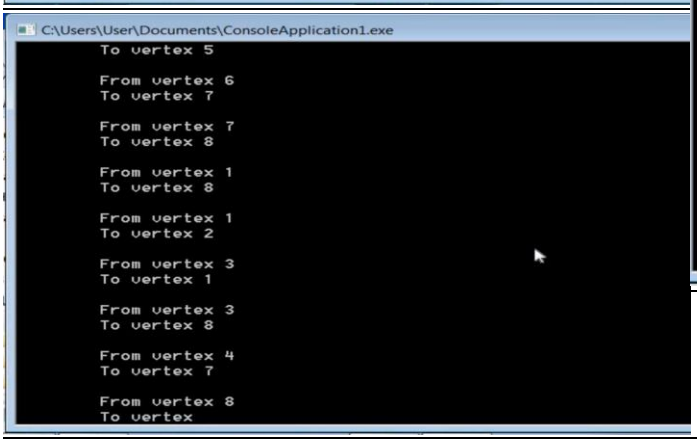

C:\Users\User\Documents\ConsoleApplication1.exe
From vertex 4
To vertex 7
From vertex 8
To vertex 9
From vertex 9
To vertex 10
From vertex 10
To vertex 8
From vertex


C:\Users\User\Documents\ConsoleApplication1.exe
Number of vertices, N = 10
Number of edges, M = 17
Enter edges
From vertex 2
To vertex 3


C:\Users\User\Documents\ConsoleApplication1.exe
To vertex 10
From vertex 10
To vertex 8
From vertex 8
To vertex 7

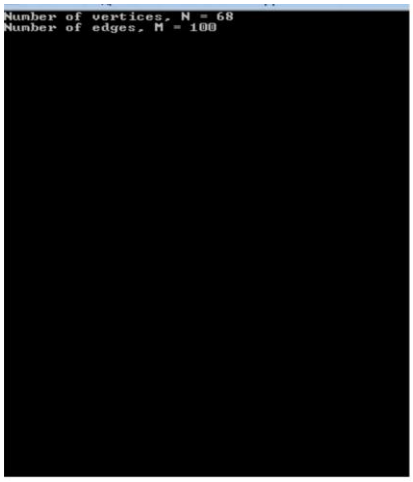

C:\Users\User\Documents\ConsoleApplication1.exe
Number of vertices, N = 10
Number of edges, M = 17
Enter edges
From vertex 2
To vertex 3
From vertex 3
To vertex 4
From vertex 4
To vertex 5
From vertex 5
To vertex 4
From vertex 5
To vertex 6
From vertex 6
To vertex 5
From vertex 6
To vertex 7

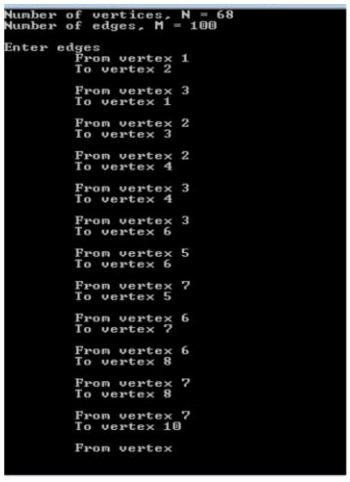

C:\Users\User\Documents\ConsoleApplication1.exe
To vertex 10
From vertex 10
To vertex 8
From vertex 8
To vertex 7
SCC list
1 3 2
4 5 6
8 7 10 9

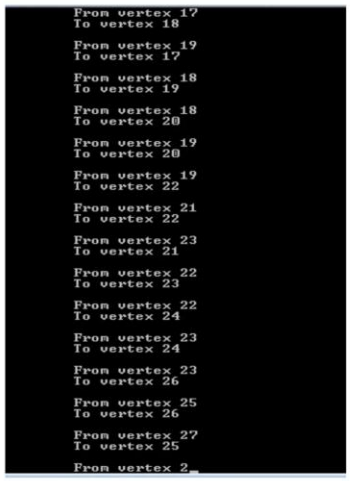

C:\Users\User\Documents\ConsoleApplication1.exe
To vertex 5
From vertex 6
To vertex 7
From vertex 7
To vertex 8
From vertex 1
To vertex 8
From vertex 1
To vertex 2
From vertex 3
To vertex 1
From vertex 3
To vertex 8
From vertex 4
To vertex 7
From vertex 8
To vertex

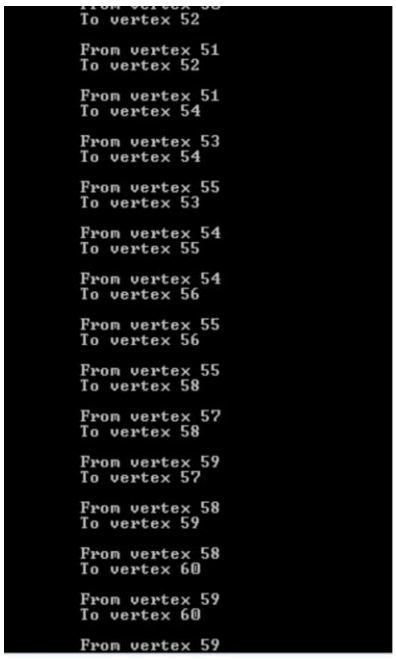
4)

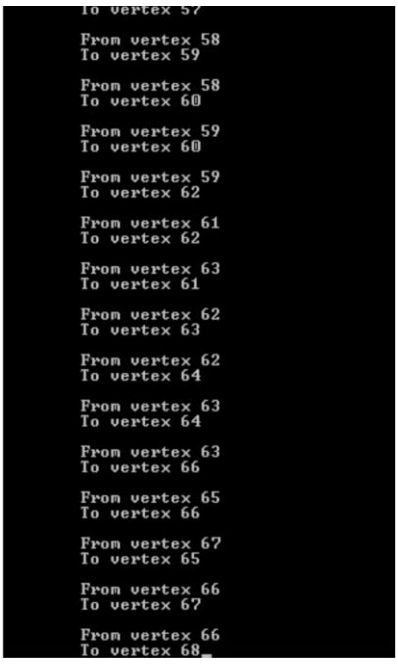
Найдем для примера SCC у графа со ста дугами.

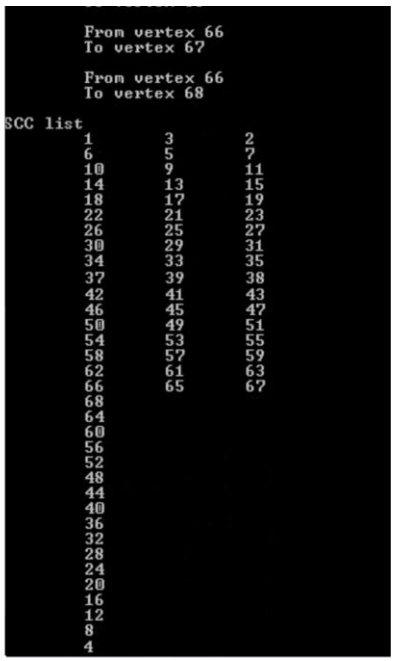







To vertex 52
From vertex 51
To vertex 52
From vertex 51
To vertex 54
From vertex 53
To vertex 54
From vertex 55
To vertex 53
From vertex 54
To vertex 55
From vertex 54
To vertex 56
From vertex 55
To vertex 56
From vertex 55
To vertex 58
From vertex 57
To vertex 58
From vertex 59
To vertex 57
From vertex 58
To vertex 59
From vertex 58
To vertex 60
From vertex 59
To vertex 60
From vertex 59


From vertex 58
To vertex 59
From vertex 58
To vertex 60
From vertex 59
To vertex 60
From vertex 59
To vertex 62
From vertex 61
To vertex 62
From vertex 63
To vertex 61
From vertex 62
To vertex 63
From vertex 62
To vertex 64
From vertex 63
To vertex 64
From vertex 63
To vertex 66
From vertex 65
To vertex 66
From vertex 67
To vertex 65
From vertex 66
To vertex 67
From vertex 66
To vertex 68


From vertex 66
To vertex 67
From vertex 66
To vertex 68
SCC list
1 3 2
6 5 7
10 9 11
14 13 15
18 17 19
22 21 23
26 25 27
30 29 31
34 33 35
37 39 38
42 41 43
46 45 47
50 49 51
54 53 55
58 57 59
62 61 63
66 65 67
68
64
60
56
52
48
44
40
36
32
28
24
20
16
12
8
4

Заключение

Так как зависимость времени работы алгоритма асимптотически линейна от размера графа (так утверждается в книге), ожидать большой трудоемкости при обработке графа большей размерности не следует.

Список литературы

[1] Седжвик Р. Фундаментальные алгоритмы на C++. Алгоритмы на графах. – СПб: ООО «ДиаСофтЮП», 2002. — 496 с.

[2] Ахо А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы. – М.: Издательский дом «Вильямс», 2000. – 384 с.

[3] Богомолов А.М. Алгебраические основы теории дискретных систем. – М.: Наука,1997. — 368 с.

[4] Кормен Т. Алгоритмы: построение и анализ. – М.: Издательский дом «Вильямс», 2013. – 1328 с.